# Коректност на рекурсивни алгоритми

Тодор Дуков

## Примери за рекурсивни алгоритми

Нека разгледаме следният алгоритъм:

```
int maximum(int *arr, int n)
{
    if (n == 0)
        return -INF;
    else
        return max(arr[n - 1], maximum(arr, n - 1));
}
```

Очевидно при параметри масив $\texttt{arr}$ с размер поне $\texttt{n}$, функцията $\texttt{maximum}(\texttt{arr}, \texttt{n})$ връща $\max \texttt{arr}[0 \dots \texttt{n} - 1]$. Ще докажем това с индукция по $\texttt{n}$:

- за $\texttt{n} = 0$ имаме, че:

$$\texttt{maximum}(\texttt{arr}, \texttt{n}) = -\infty = \max[] = \max \texttt{arr}[0 \dots 0 - 1], \text{ където под } [] \text{ се има предвид празният масив}$$

- за $\texttt{n} + 1$ имаме, че:

$$\texttt{maximum}(\texttt{arr}, \texttt{n} + 1) = \max(\texttt{arr}[\texttt{n}], \texttt{maximum}(\texttt{arr}, \texttt{n})) \overset{\text{(ИП)}}{=} \max(\texttt{arr}[\texttt{n}], \max \texttt{arr}[0 \dots \texttt{n} - 1]) = \max \texttt{arr}[0 \dots \texttt{n}]$$

Тук управляващият параметър на рекурсията $\texttt{n}$ винаги намалява с $1$, докато не стигне $0$, където ще приключи алгоритъмът. В по нататъчните разсъждения ще смятаме това за очевидно.
Сложността на алгоритъма се описва със рекурентното уравнение:

$$T(n) = T(n-1) + 1 \;//\; \text{базата няма да я пишем}$$

Директно се вижда, че $T(n) = \sum\limits_{i=0}^{n} 1 = n + 1 \asymp n$.

Да видим един малко по-сложен пример – за бързо степенуване:

```
int exp(int base, int power)
{
    if (power == 0)
        return 1;

    int small = exp(base, power / 2);

    if (power % 2 == 1)
        return small * small * base;
    else
        return small * small;
}
```

С пълна (от тук нататък няма да го пишем) индукция относно $\texttt{power}$ ще покажем, че $\texttt{exp}(\texttt{base}, \texttt{power}) = \texttt{base}^{\texttt{power}}$:

- $\texttt{exp}(\texttt{base}, 0) = 1 = \texttt{base}^0$ // тук се уговаряме, че $0^0 = 1$
- $\texttt{exp}(\texttt{base}, 2\texttt{n} + 1) = \texttt{base} \cdot \texttt{exp}(\texttt{base}, \texttt{n}) \cdot \texttt{exp}(\texttt{base}, \texttt{n}) \overset{\text{(ИП)}}{=} \texttt{base} \cdot \texttt{base}^{\texttt{n}} \cdot \texttt{base}^{\texttt{n}} = \texttt{base}^{2\texttt{n}+1}$
- $\texttt{exp}(\texttt{base}, 2\texttt{n} + 2) = \texttt{exp}(\texttt{base}, \texttt{n} + 1) \cdot \texttt{exp}(\texttt{base}, \texttt{n} + 1) \overset{\text{(ИП)}}{=} \texttt{base}^{\texttt{n}+1} \cdot \texttt{base}^{\texttt{n}+1} = \texttt{base}^{2\texttt{n}+2}$

Сложността на този алгоритъм може да се опише със следното рекурентно уравнение:

$$T(n) = T(\frac{n}{2}) + 1$$

От мастър-теоремата следва, че:

$$T(n) \asymp \log(n)$$

# Трик за бързо пресмятане на членове на някои рекурентни редици

Нека вземем за пример редицата на Фибоначи:

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(n+2) &= F(n+1) + F(n) \end{aligned}$$

Човек може да забележи, че:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} F(n+1) \\ F(n) \end{pmatrix} = \begin{pmatrix} F(n+2) \\ F(n+1) \end{pmatrix}$$

След това с индукция лесно се показва, че:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \cdot \begin{pmatrix} F(1) \\ F(0) \end{pmatrix} = \begin{pmatrix} F(n+1) \\ F(n) \end{pmatrix}$$

За да сметнем $F(n)$, можем да направим бързо степенуване на матрицата, аналогична на тази с числата:

```
void multiply(int A[2][2], int B[2][2]) // записва резултата в A
{
    int res[2][2];
    res[0][0] = A[0][0] * B[0][0] + A[0][1] * B[1][0];
    res[0][1] = A[0][0] * B[0][1] + A[0][1] * B[1][1];
    res[1][0] = A[1][0] * B[0][0] + A[1][1] * B[1][0];
    res[1][1] = A[1][0] * B[0][1] + A[1][1] * B[1][1];

    for (int i = 0; i < 2; ++i)
        for (int j = 0; j < 2; ++j)
            A[i][j] = res[i][j];
}

void fib_matrix_exp(int result[2][2], int n)
{
    if (n == 0)
    {
        result[0][0] = result[1][1] = 1;
        result[0][1] = result[1][0] = 0;
        return;
    }

    fib_matrix_exp(result, n / 2);
    multiply(result, result);

    if (n % 2 == 1)
    {
        int fib_matrix[2][2] = {{1, 1},
                                {1, 0}};
        multiply(result, fib_matrix);
    }
}

int fib(int n)
{
    if (n <= 1)
        return n;

    int matrix[2][2];
    fib_matrix_exp(matrix, n - 1);

    return matrix[0][0];
}
```

Коректността и сложността на алгоритъма оставяме на читателя (напълно аналогично е на предния алгоритъм).

В общия случай ще имаме рекурентно уравнение от вида:

$$T(n+k+1) = a_k T(n+k) + \dots + a_1 T(n+1) + a_0 T(n), \text{ където } a_0, \dots, a_k, k \text{ са константи}$$

Тогава имаме следната зависимост:

$$\begin{pmatrix} a_k & a_{k-1} & a_{k-2} & \dots & a_0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} T(n+k) \\ T(n+k-1) \\ T(n+k-2) \\ \vdots \\ T(n) \end{pmatrix} = \begin{pmatrix} T(n+k+1) \\ T(n+k) \\ T(n+k-1) \\ \vdots \\ T(n+1) \end{pmatrix}$$

Отново с индукция лесно се показва, че:

$$\begin{pmatrix} a_k & a_{k-1} & a_{k-2} & \dots & a_0 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & 0 \end{pmatrix}^n \cdot \begin{pmatrix} T(k) \\ T(k-1) \\ T(k-2) \\ \vdots \\ T(0) \end{pmatrix} = \begin{pmatrix} T(n+k) \\ T(n+k-1) \\ T(n+k-2) \\ \vdots \\ T(n) \end{pmatrix}$$

## Задачи

*Задача* 1. Да се напише алгоритъм $\texttt{calculate}(\texttt{first\_members}, \texttt{step}, \texttt{k}, \texttt{n})$, който приема два масива от цели числа `first_members` и `step` с размер $\texttt{k}+1$, естествено число `n`, и връща `n`-тия член на следната рекурентна редица:

$$\begin{aligned} T(0) &= \texttt{first\_members}[0] \\ T(1) &= \texttt{first\_members}[1] \\ &\vdots \\ T(k) &= \texttt{first\_members}[\texttt{k}] \\ T(n+k+1) &= \texttt{step}[\texttt{k}] \cdot T(n+k) + \dots + \texttt{step}[1] \cdot T(n+1) + \texttt{step}[0] \cdot T(n) \end{aligned}$$

След това да се докаже неговата коректност, и да се изследва сложността му по време.

*Задача* 2. Даден е следният алгоритъм:

```
void stooge_sort(int *arr, int l, int h)
{
    if (l >= h)
        return;

    if (arr[l] > arr[h])
    {
        int temp = arr[l];
        arr[l] = arr[h];
        arr[h] = temp;
    }

    int t = (h - l + 1) / 3;

    if (t >= 1)
    {
        stooge_sort(arr, l, h - t);
        stooge_sort(arr, l + t, h);
        stooge_sort(arr, l, h - t);
    }
}
```

Да се докаже, че при подаден масив `arr` с размер `n`, функцията $\texttt{stooge\_sort}(\texttt{arr}, 0, \texttt{n}-1)$ ще сортира `arr`. След това да се направи анализ на сложността по време.